Hardware/Software Co-Design Flavors of Elliptic Curve Scalar Multiplication

Josep Balasch*, Benedikt Gierlichs*, Kimmo Järvinen*†, and Ingrid Verbauwhede*
*KU Leuven, ESAT/COSIC and iMinds
Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
†Aalto University, Department of Information and Computer Science
Konemiehentie 2, 02150 Espoo, Finland
Email: firstname.lastname@esat.kuleuven.be

***Abstract*—Many electronic applications use cryptographic algorithms implemented in embedded devices to provide some form of security, e.g. smart cards (banking, SIM, access control), mobile phones, wifi routers, etc. The tight resource constraints of the devices, typically silicon area and power or energy, together with requirements from the application, typically latency or throughput, demand highly efficient implementations of the often computationally complex cryptographic algorithms. We provide a broad overview of the hardware/software co-design space for an essential component of many cryptographic protocols. Based on our experience from teaching a master level course about hardware/software co-design, we explore four typical implementation options and provide concrete implementation results. In addition to the aforementioned criteria, resistance against implementation attacks is vital for the security of embedded cryptographic devices. We analyze our four implementations with respect to a security issue that is due to their electromagnetic emanations, and highlight multiple vulnerabilities that can be exploited to break their security. Next, we investigate state-of-the-art implementation options that are supposed to resist these attacks. We detail their implementation cost and show that it is non-trivial to implement these options securely. Our main contribution is a comprehensive analysis of many implementation options with respect to implementation cost and attack resistance on a single common platform.**

I. Introduction

The *Design of Digital Platforms* is a master level course that introduces students to the topic of *hardware/software co-design*. The course is composed of theory-oriented lectures and project-oriented sessions. Lectures cover aspects of data-flow and control-flow analysis, impact of memory management and design methodologies. Practical sessions provide insight and hands-on experience by developing a public-key cryptographic system [1]. This paper focuses on the project-oriented sessions. We select Elliptic Curve Cryptography (ECC) [2], [3] as exemplary hardware/software co-design application, as previously done in [4]. Past editions of the course used other cryptographic systems such as RSA [5].

An elliptic curve E over a prime field $\mathbb{F}_p$ with $p > 3$ can be defined by the short *Weierstrass* equation $y^2 = x^3 + ax + b$. Here, a and b are preset parameters of the curve. The affine coordinates (x, y) represent a point P on the elliptic curve. An alternative representation widely used in practice is given by projective coordinates. Here a point P is represented by a tuple (X, Y, Z), where $x = X/Z^2$ and $y = Y/Z^3$. In the course we employ a NIST prime-field curve [6] P-256 over $\mathbb{F}_p$ where $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$.

Because of its mathematical and modular nature, an ECC implementation can be developed by following a bottom-up approach as depicted in Figure 1 (left). This characteristic allows one to divide project sessions into multiple assignments that build on top of each other. The first assignments correspond to the lower *field arithmetic* layer. Here, students have to implement operations over the prime field $\mathbb{F}_p$, namely, field addition, field subtraction and field multiplication. After this, students move on to the implementation of the *point arithmetic* layer, which include two operations: point addition ($P_3 = P_1 + P_2$) and point doubling ($P_3 = 2P_1$). We consider three different algorithms to compute these operations: two for point addition denoted as `PADD_JJJ` and `PADD_JJA`, and one for point doubling denoted by `PDBL_JJ`. Here, we use J (Jacobian) and A (affine) to refer to the coordinate representation of P_3, P_1, and P_2, respectively. Finally, upon completion of this step, students arrive at the final assignment of the course: the implementation of ECC *scalar multiplication*.

The implementation of the lower field arithmetic layer is developed first in software and then in hardware. This allows students to quickly grasp the obvious limitations of hardware-only and software-only solutions, i.e. software results in slow but flexible implementations, while hardware achieves fast designs at higher area costs but is unupdatable. These observations are extensively used in the remaining practical sessions. The end goal of the project is to obtain an implementation that optimizes the typical co-design trade-off between area, speed and flexibility, as illustrated in Figure 1 (right).

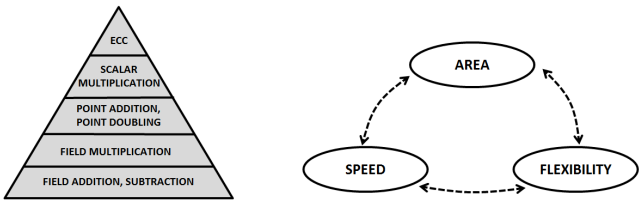

Fig. 1. Implementation layers of ECC (left), co-design trade-off (right).

The algorithms to perform ECC scalar multiplication, point arithmetic and field arithmetic are fixed at the beginning of the course to allow a fair comparison between students' designs. Regarding the final assignment, students are asked

to implement the double-and-add binary scalar multiplication depicted in Alg. 1. Given a point $P \in \mathbb{F}_p$ and a scalar k, the algorithm computes a new point $Q \in \mathbb{F}_p$ of the form $Q = kP$. The structure of Alg. 1 is rather simple. It iterates over each bit of the input scalar k and performs at most two point operations per iteration: a point doubling PDBL_JJ followed by a conditional point addition PADD_JJA when k_i equals one.

Algorithm 1 Double-and-add (left-to-right).

INPUT: Base point $P \in E(\mathbb{F}_p)$, scalar $k = (k_{t-1}, \ldots, k_0)_2$
OUTPUT: Point $Q = k \cdot P$
  $R_0 \leftarrow \infty$; $R_1 \leftarrow P$
  for $i = t - 1$ to 0 **do**
    $R_0 \leftarrow$ PDBL_JJ(R_0)
    if $k_i == 1$ **then**
      $R_0 \leftarrow$ PADD_JJA(R_0, R_1)
    end if
  end for
  $Q = R_0$

In order to allow students to explore the full co-design space, we select a representative target platform composed of an 8-bit microcontroller (i.e. software) attached to a customizable arithmetic co-processor (i.e. hardware). The popular 8051 architecture is selected as embedded microcontroller. Its four 8-bit parallel ports are used to interface with the arithmetic co-processor and to enable a shared memory component, as illustrated in Figure 2. The MCU 8051 Integrated Development Environment [7] is used for software development, while the GEZEL [8], [9] description language is used for hardware development and co-simulation.

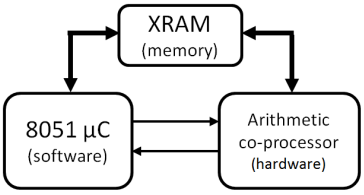

Fig. 2. Architecture for target platform.

The combination of hardware and/or software solutions with the implementation granularity of the ECC arithmetic pyramid gives rise to multiple designs and to multiple trade-off options. In general, however, the vast majority of students in the course end up with one of the following partitions between hardware and software:

1) Co-processor supports field multiplication.
2) Co-processor supports field arithmetic (addition, subtraction and multiplication).
3) Co-processor supports point arithmetic (point doubling, point addition).
4) Co-processor supports scalar multiplication.

A. Structure and contributions

In the rest of the paper we evaluate each of these hardware/software partitions. In Section II we describe our own implementations of four co-processor implementations. Each of these designs is representative of the implementations achieved by students, and as such it maps to a particular generic co-design boundary. We provide a detailed evaluation of the cost of all implementations by implementing them on an FPGA. This analysis follows the same criteria as in the course, i.e. focusing on area, speed and flexibility. In Section III we add a fourth dimension criteria in the analysis of the designs, namely security against electromagnetic side-channel attacks. By means of practical experiments, we highlight multiple vulnerabilities found in these generic designs. We further analyze the role of the hardware/software partition in each vulnerability. In Section IV we introduce techniques from the literature that aim to solve these issues. They are based on algorithms for ECC scalar multiplication with constant patterns of point operations. We demonstrate that while these algorithms are indeed suited to counter electromagnetic side-channel attacks, their secure implementation is non-trivial. Finally, we conclude in Section V.

II. THE CO-PROCESSORS

We prepared four different arithmetic co-processors according to the common partitions used by the students. We prepared these co-processors so that they are modular in the sense that the next co-processor supports all functionalities of the previous one. Implementation details of our co-processors are described in the following subsections together with their implementation results on a Xilinx Virtex-5 FPGA.

A. Co-processor 1 (CP 1)

The simplest co-processor (CP 1) provides support for field multiplication. The finite field multiplier of CP 1 computes N-bit multiplications, where $N = 256$, using a $W \times N$-bit multiplier and a $N + W$-bit adder that accumulates a $2N$-bit shift register. Let $B(i)$ denote the ith W-bit word of B: $B = \sum_{i=0}^{N/W-1} B(i)2^{Wi}$. The operands are fed to the multiplier one word per clock cycle. The entire element A is first loaded into the multiplier using a W-bit interface. Then, each $B(i)$ is loaded starting from $B(0)$ and multiplied by A using the $W \times N$-bit multiplier. The accumulator updates the least significant $N + W$ bits of the shift register and shifts the register by W bits. When all $B(i)$ have been processed, the shift register contains the 512-bit result of the integer multiplication so that the most significant $N-W$ bits contain the least significant bits and the least significant $N+W$ bits contain the most significant bits of the result. The result of the integer multiplication is reduced modulo p using the formulae given in [6]. The result of the modular multiplication is returned one W-bit word per clock cycle starting from the least significant word. The latency of a multiplication is $3N/W + 3$ clock cycles. CP 1 reads operands from and writes results to the XRAM using the 8-bit interface of the XRAM. Hence, the logical choice for the word size is $W = 8$ bits. The 8×256-bit multiplier was implemented using hardwired multipliers (DSP blocks) available on Xilinx Virtex-5 FPGAs. Field multiplications and the interface are governed by finite state machines.

B. Co-processor 2 (CP 2)

The next co-processor (CP 2) adds support for field additions and subtractions. The finite field adder/subtractor computes additions and subtractions W bits per clock cycle. Let

TABLE I. AREAS AND MAXIMUM CLOCK FREQUENCIES OF THE FOUR CO-DESIGN PARTITIONS.

Design	Slices	LUTs	Flip-Flops	DSP48Es	BlockRAMs	Max. freq. (MHz)
μC + CP 1	2010	6457	1915	26	6	39.97
μC + CP 2	2237	7268	2470	26	6	39.01
μC + CP 3	2392	7556	2603	27	6	38.93
μC + CP 4	2525	8015	2896	27	6	39.36

$A(i)$ denote the ith W-bit word of A. The elements are fed to the adder/subtractor one word per clock cycle as follows: $A(0)$, $B(0)$, $A(1)$, $B(1)$, ..., $A(N/W-1)$, and $B(N/W-1)$. Each clock cycle the adder/subtractor computes $A(i) + B'(i) + c$, where c is the carry from the previous iteration (initialized with zero for additions and one for subtractions) and $B'(i) = B(i)$ for additions and $B(i) = \bar{B}(i)$ (all bits of $B(i)$ flipped) for subtractions, and stores the result in an N-bit shift register. Modular reduction is performed so that either p or $-p$ is added to the $N+1$-bit result for subtraction and addition, respectively, after which either the original result or the result of the addition is selected depending on their signs. The result of a modular addition or subtraction is returned one W-bit word per clock cycle starting from the least significant word. The latencies of addition and subtraction are both $3N/W + 2$ clock cycles. Similarly to CP 2, the interface of the XRAM dictates that $W = 8$ also for CP 2. The execution of field operations and the interface are controlled by finite state machines.

C. Co-processor 3 (CP 3)

As mentioned both CP 1 and CP 2 operate directly using the XRAM so that they fetch the operands from the XRAM in the beginning of each operation and write the result to the XRAM immediately after the end of the operation. As shown above, the latencies of multiplications, additions, and subtractions are almost completely determined by the width W of their interface. In the case of CP 1 and CP 2, the performance was bounded by the 8-bit interface of the XRAM and, hence, the only logical choice was $W = 8$. Therefore, the latencies of the field operations were relatively long.

The next co-processor (CP 3) solves this limitation by computing entire point operations on the co-processor. CP 3 adds an internal register file for storing intermediate values and, consequently, it largely avoids the limits of the 8-bit XRAM interface if several field operations are computed with values read from the XRAM once. In addition to field multiplications, additions, and subtractions, CP 3 is further capable of computing point operations so that it fetches the operand points P_1 and P_2 from the XRAM, computes the point operation, and writes the result point P_3 to the XRAM. Hence, CP 3 is less bounded by the 8-bit interface of the XRAM and a larger W can be used in the field multiplier resulting in significantly faster multiplications (we used $W = 32$ for CP 3). Also for CP 3, the 32×256-bit multiplier was implemented using hardwired multipliers (DSP blocks) available on Xilinx Virtex-5 FPGAs.

CP 3 supports all three algorithms for point operations using Jacobian coordinates: `PADD_JJJ`, `PADD_JJA`, and `PDBL_JJ`. Field squarings required by the algorithms are computed using the field multiplier and multiplications by small constants are computed with the adder (e.g., $X_1 \leftarrow 4 \times X_1$ is computed $X_1 \leftarrow X_1 + X_1$ followed by $X_1 \leftarrow X_1 + X_1$). The algorithms can be implemented with ten N-bit registers: X_1, Y_1, Z_1, X_2, Y_2, Z_2, T_1, T_2, T_3, and T_4. All ten registers are used only by `PADD_JJJ`. The 10×256-bit register file was implemented using distributed RAM. The algorithms were derived using the formulae available in the online Explicit-Formulas Database [10]. Note that the Alg. 1 is implemented with `PADD_JJA` and `PDBL_JJ`. The reason why CP 3 includes also support for `PADD_JJJ` is due to security related improvements which are discussed later in Sect. III.

D. Co-processor 4 (CP 4)

The last co-processor (CP 4) adds hardware support also for scalar multiplication algorithms and, consequently, removes the need for software control during scalar multiplication. CP 4 fetches the scalar k and the generator point P from the XRAM, stores them to the register file, computes $Q = kP$ using one of the supported algorithms, and writes Q to the XRAM. Hence, the amount of data that needs to be transferred between CP 4 and the XRAM using the slow 8-bit interface is very small.

The students' solutions include support only for Alg. 1. However, because of security related considerations discussed later in the paper, we implemented CP 4 to support three algorithms for scalar multiplication: Alg. 1, Alg. 2, and Alg. 3. Point operations and scalar multiplication algorithms are executed by using finite state machines. Both Alg. 1 and Alg. 3 can be implemented with the ten registers available in the register file of CP 3. However, Alg. 2 requires three additional registers for the dummy point R_{-1}. Hence, the register file of CP 4 includes 13 N-bit registers. Also this 13×256-bit register file was implemented using distributed RAM.

E. Performance

All designs were described in VHDL and compiled for Xilinx Virtex-5 xc5vlx30-2ff324 FPGA using Xilinx ISE 13.4 software. Table I collects the area and performance results (after place&route). As can be seen, the growth in area was moderate between the designs because most of the resources are used by the multiplier. The register files were implemented using distributed RAM which significantly decreased their resource requirements. The use of a larger W in CP 3 and CP 4 resulted in an only relatively small increase in area requirements. Regardless of the implementation, the area requirements for the embedded microcontroller were constant and amounted to 1 101 slices and 500 flip flops. These numbers are already included in the results of Table I.

Latencies of different operations supported by the co-processors (in clock cycles) are listed in Table II. They include the interfacing with the XRAM but not delays depending on the software. In other words, they represent the optimal case in which the software side has no overhead on the overall computation. The dash (—) indicates that the operation is not supported by the co-processor. The latencies are accompanied by either C or V depending whether the latency is constant

TABLE II. LATENCIES OF OPERATIONS SUPPORTED BY THE CO-PROCESSORS.

Co-processor	Multiplication	Addition	Subtraction	PDBL_JJ	PADD_JJA	PADD_JJJ	Alg. 1	Alg. 2	Alg. 3
CP 1	100 (C)	—	—	—	—	—	—	—	—
CP 2	100 (C)	99 (C)	99 (C)	—	—	—	—	—	—
CP 3	147 (C)	146 (C)	146 (C)	899 (C)	801 (C)	1,148 (C)	—	—	—
CP 4	147 (C)	146 (C)	146 (C)	899 (C)	801 (C)	1,148 (C)	240,000 (V)	306,000 (C)	387,000 (C)

(C) constant, (V) varying

or varying depending on the value of the operands and/or the secret scalar k. CP 3 and CP 4 compute field operation instructions slightly slower than CP 1 and CP 2 because the elements are first written into the register file. For the scalar multiplication algorithms, it is assumed that k is a 256-bit integer where half of the bits are ones; the latency of scalar multiplication is considered constant if it does not depend on the number of ones in the scalar k.

III. SECURITY EVALUATION

In addition to the criteria area, speed and flexibility, resistance against side-channel attacks is vital for embedded applications of cryptography. It is well known that straightforward implementations without explicit consideration of this security aspect can be easily broken by a variety of attacks [11]. Here we focus on attacks using simple electromagnetic analysis [12], [13].

For our security evaluation, we implemented each of the four designs (μC and co-processor) in a Xilinx Spartan3E-500 FPGA on a development board and clocked it at 24 MHz frequency. We placed the board on a X-Y-Z positioning system and attached it with a custom jig to prevent any movement of the board relative to the positioning system. We used a magnetic near field probe [14] (500 μm diameter, horizontal orientation) to capture electromagnetic emanations above the upper surface of the FPGA. This type of measurement setup is common in EMC testing.

We positioned the probe less than 50 μm above the FPGA package to obtain a good signal strength. We conditioned the signal with a built-in pre-amplifier, a 48 MHz low-pass filter and an additional amplifier (30 dB) before sampling it with a digital storage oscilloscope at a rate of 125 MS/s. In contrast to EMC testing, where one is usually interested in the frequency spectrum, we are interested in a time signal.

Figure 3 shows a plot of the field amplitude over time (EM trace) measured while the first implementation with CP 1 performs an elliptic curve scalar multiplication. We can clearly see when the operation begins (the amplitude increases) and when it ends (the amplitude drops). We can further observe that it takes about 400 ms (approx. 10^7 cycles) to complete the operation.

Figure 4 shows a zoom of the same EM trace. It covers the first few point operations. The field multiplier constitutes roughly 50% of the FPGA resources used in this implementation, and it performs a multiplication in short time compared to addition and subtraction that are implemented in software. This leads to high, fast changing currents and hence relatively strong emissions. In the figure, the horizontal "band" represents the microcontroller's activity whereas the vertical peaks that exceed the "band" are due to the multiplier's activity.

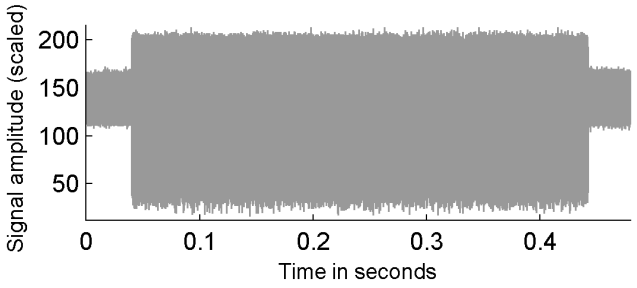

Fig. 3. EM trace for double-and-add algorithm, implementation with CP 1, full execution.

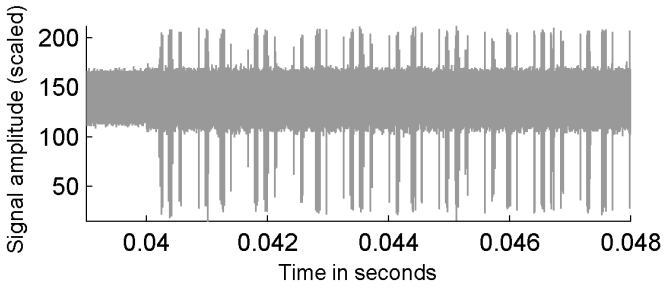

Fig. 4. EM trace for double-and-add algorithm, implementation with CP 1, execution of a few point operations.

Figure 5 shows an even more detailed zoom of two EM traces (plotted in different shades of gray). The two traces were captured during scalar multiplications of different input points P. They are perfectly aligned at the beginning of the operation, but they deviate from each other rather soon. The figure shows a short sequence of four field multiplications that occurs in both scalar multiplications, but at different time offsets.

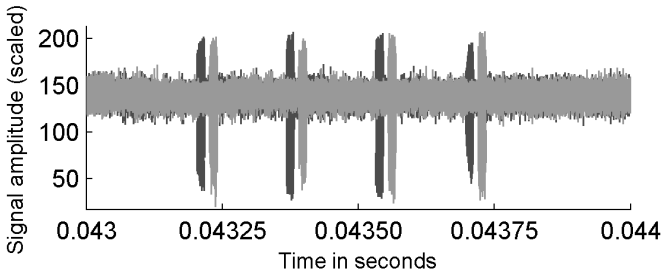

Fig. 5. EM trace for double-and-add algorithm, implementation with CP 1, execution of a few field operations.

The timing differences are deterministic and caused by the non-constant time software implementations of field addition and subtraction. Both these operations need to ensure that the output value is a valid element in $\mathbb{F}_p$, i.e. an integer in the interval $[0, p-1]$. Ensuring this requires the execution

of a conditional operation to obtain the correct result, giving rise to timing variations depending on the data processed. An adversary can exploit these differences and potentially recover the full secret scalar from a single EM trace. It is therefore crucial to address this problem, for instance by designing implementations that execute in constant time.

The second implementation uses CP 2 that provides all field arithmetic in constant time hardware. Figure 6 shows an EM trace of this implementation. A first observation is that this implementation requires about 120ms (approx. 3×10^6 cycles) to complete the scalar multiplication.

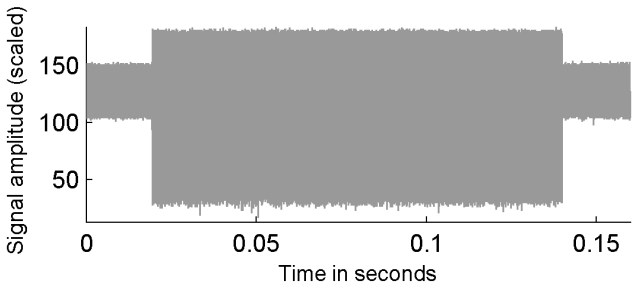

Fig. 6. EM trace for double-and-add algorithm, implementation with CP 2, full execution.

We verified that the field arithmetic in CP 2 indeed executes in constant time. However, there are more security issues. Figure 7 shows a zoom on the first four point operations in the same EM trace. The field multiplier still constitutes roughly 50% of the FPGA resources in the implementation and its activity causes the same clearly visible peaks.

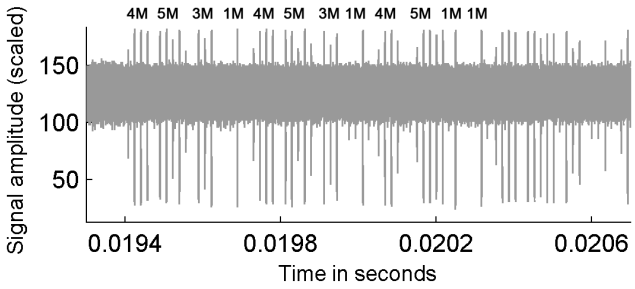

Fig. 7. EM trace for double-and-add algorithm, implementation with CP 2, execution of first four point operations.

The sequences of field operations for point doublings and point additions are similar but not identical. Let `M` and `A` denote field multiplication and field addition or subtraction, respectively. The number and sequence of field operations for each of these operations is:

`PDBL_JJ`: $\texttt{4M} \rightarrow \texttt{1A} \rightarrow \texttt{5M} \rightarrow \texttt{1A} \rightarrow \texttt{3M} \rightarrow \texttt{3A} \rightarrow \texttt{1M} \rightarrow \texttt{1A}$
`PADD_JJA`: $\texttt{4M} \rightarrow \texttt{2A} \rightarrow \texttt{5M} \rightarrow \texttt{1A} \rightarrow \texttt{1M} \rightarrow \texttt{3A} \rightarrow \texttt{1M} \rightarrow \texttt{1A}$

The figure shows that the number and sequence of peaks in the EM trace directly corresponds to the number and sequence of field multiplications. An adversary can exploit this correspondence to recover the sequence of point operations, which immediately leads to the secret scalar (recall Alg. 1), potentially from a single EM trace. Implementations three and four (with CP 3 and CP 4, respectively) require substantially less time to process point operations and a scalar multiplication, but both suffer from the same vulnerability since they also use the double-and-add algorithm.

IV. Regular scalar multiplication algorithms

The literature provides two popular alternatives to the double-and-add algorithm: the double-and-add-always algorithm (see Alg. 2) and the Montgomery Ladder (see Alg. 3).

Algorithm 2 Double-and-add-always (left-to-right) [15].

INPUT: Base point $P \in E(\mathbb{F}_p)$, scalar $k = (k_{t-1}, \ldots, k_0)_2$
OUTPUT: Point $Q = k \cdot P$
  $R_0 \leftarrow \infty$; $R_1 \leftarrow P$
  for $i = t - 1$ to 0 **do**
    $b \leftarrow k_i$
    $R_0 \leftarrow \texttt{PDBL\_JJ}(R_0)$
    $R_{-b} \leftarrow \texttt{PADD\_JJA}(R_{-b}, R_1)$
  end for
  $Q = R_0$

Algorithm 3 Montgomery Ladder (left-to-right) [16].

INPUT: Base point $P \in E(\mathbb{F}_p)$, scalar $k = (k_{t-1}, \ldots, k_0)_2$
OUTPUT: Point $Q = k \cdot P$
  $R_0 \leftarrow \infty$; $R_1 \leftarrow P$
  for $i = t - 1$ to 0 **do**
    $b \leftarrow k_i$
    $R_{1-b} \leftarrow \texttt{PADD\_JJJ}(R_{1-b}, R_b)$
    $R_b \leftarrow \texttt{PDBL\_JJ}(R_b)$
  end for
  $Q = R_0$

As discussed in Sect. III, Alg. 1 employs a pattern of point operations dependent on k and, hence, it leaks information about the secret value for an adversary who measures e.g. the EM characteristics of the implementation when computing scalar multiplications. Both Alg. 2 and Alg. 3, however, have a constant pattern of point operations which significantly improves their security against side-channel attacks. Nevertheless, it is still not easy to implement these algorithms securely.

Figure 8 shows a zoom of two EM traces (plotted in different shades of gray) that were captured during scalar multiplications with two different keys K_1 and K_2. The implementation uses CP 2 and the double-and-add-always algorithm in software. K_1 is equal to K_2 except for the first byte. The first byte of K_1 is $1000\,0000_b$ while the first byte of K_2 is $1000\,1000_b$. The figure shows a part of the point addition computation during the processing of the key bit that differs, i.e. k_{t-5}. We can observe a small but noticeable time shift between both traces. Before the processing of this key bit both traces are perfectly aligned, and the time shift is constant during the processing of the remaining key bits. This time shift is deterministic and caused by if-then-else statements in the software that determine if the point addition is real or dummy. Hence, although the scalar multiplication algorithm has a constant pattern of point operations, an adversary would still be able to determine the value of the secret scalar, possibly from a single EM trace, due to such conditional branches. A secure implementation should therefore not use conditional branches, which is not easy in particular in software.

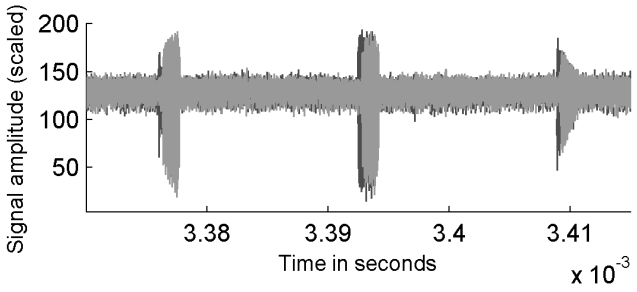

Fig. 8. EM trace for double-and-add-always algorithm, implementation with CP 2.

Although all of our co-processors are insecure if scalar multiplications are computed using the double-and-add algorithm (Alg. 1), they all offer features that enable secure implementations if combined with scalar multiplication algorithms that employ a constant pattern of point operations. The supported field operations (multiplications, additions, and subtractions) are always computed with constant latency which nullifies efforts to determine the secret scalar by observing varying latencies of field operations depending on the known generator point P. Furthermore, all point operations supported by CP 3 and CP 4 are computed with constant latencies (i.e., the latencies do not depend on the values of the input points P_1 and P_2 or the secret scalar k). However, an adversary may still observe which particular point operation is being computed because the pattern of field operations is different for different point operations as explained in Sect. III. Hence, the point operations offer side-channel resistance only when employed by a scalar multiplication algorithm that has a constant pattern of point operations such as Alg. 2 or Alg. 3. In summary, two out of the three scalar multiplication algorithms supported by CP 4 offer strong protection against adversaries who aim to find the value of the scalar by observing a single power or EM trace.

We must, however, emphasize that even CP 4 does not offer foolproof protection against very powerful adversaries. For instance, if an adversary is able to perform very precise localized EM measurements, (s)he can be able to distinguish which registers in the register file are written to at a given point in time, which would allow breaking both Alg. 2 and Alg. 3 [17]. Also, if the adversary is able to inject faults into the computation, then (s)he can break Alg. 2 by corrupting the result of a point addition [18]. If the fault corrupts the result of the scalar multiplication, then it was injected during a real point addition ($k_i = 1$), if not, then it affected a dummy point addition ($k_i = 0$).

V. Conclusion

Elliptic curve scalar multiplication is an important building block in many cryptographic protocols. We provide a comprehensive study of many implementation options in the hardware/software co-design space focusing on the criteria area, latency and security against electromagnetic side-channel attacks. All our results stem from implementations on a single common platform, hence their comparison is meaningful. They can serve practitioners to make the right implementation choices, and in particular point out security pitfalls.

Acknowledgment

This work was supported in part by the Research Council KU Leuven: GOA TENSE (GOA/11/007) and F+ fellowship (F+/13/039). In addition, this work is supported in part by the Flemish Government, FWO G.0550.12N, G.00130.13N and FWO G.0876.14N, by the Hercules Foundation AKUL/11/19, and by Intel. Benedikt Gierlichs is an FWO postdoctoral researcher.

References

[1] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[2] V. S. Miller, "Use of Elliptic Curves in Cryptography," in *Advances in Cryptology - CRYPTO '85*, ser. LNCS, H. C. Williams, Ed., vol. 218. Springer, 1985, pp. 417–426.

[3] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.

[4] L. Batina, D. Hwang, A. Hodjat, B. Preneel, and I. Verbauwhede, "Hardware/Software Co-design for Hyperelliptic Curve Cryptography (HECC) on the 8051 uP," in *CHES 2005*, ser. LNCS, J. R. Rao and B. Sunar, Eds., vol. 3659. Springer, 2005, pp. 106–118.

[5] R. L. Rivest, A. Shamir, and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[6] NIST, Digital Signature Standard (DSS) - FIPS-186-4, 2013. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

[7] MCU 8051 Integrated Development Enviroment. [Online]. Available: http://mcu8051ide.sourceforge.net/

[8] GEZEL Hardware/Software Codesign Environment. [Online]. Available: http://rijndael.ece.vt.edu/gezel2/

[9] P. Schaumont, *A Practical Introduction to Hardware/Software Codesign*. Springer, 2010.

[10] D. J. Bernstein and T. Lange, Explicit-Formulas Database. [Online]. Available: http://www.hyperelliptic.org/EFD/

[11] P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology - CRYPTO '99*, ser. LNCS, M. J. Wiener, Ed., vol. 1666. Springer, 1999, pp. 388–397.

[12] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic Analysis: Concrete Results," in *CHES 2001*, ser. LNCS, Ç. K. Koç, D. Naccache, and C. Paar, Eds., vol. 2162. Springer, 2001, pp. 251–261.

[13] J.-J. Quisquater and D. Samyde, "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards," in *E-smart 2001*, ser. LNCS, I. Attali and T. P. Jensen, Eds., vol. 2140. Springer, 2001, pp. 200–210.

[14] Langer EMV-Technik, ICR near-field microprobes. [Online]. Available: http://www.langer-emv.com

[15] J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *CHES'99*, ser. LNCS, Ç. K. Koç and C. Paar, Eds., vol. 1717. Springer, 1999, pp. 292–302.

[16] M. Joye and S.-M. Yen, "The Montgomery Powering Ladder," in *CHES 2002*, ser. LNCS, B. S. Kaliski Jr., Ç. K. Koç, and C. Paar, Eds., vol. 2523. Springer, 2002, pp. 291–302.

[17] J. Heyszl, S. Mangard, B. Heinz, F. Stumpf, and G. Sigl, "Localized Electromagnetic Analysis of Cryptographic Implementations," in *CT-RSA 2012*, ser. LNCS, O. Dunkelman, Ed., vol. 7178. Springer, 2012, pp. 231–244.

[18] S.-M. Yen and M. Joye, "Checking before output may not be enough against fault-based cryptanalysis," *IEEE Trans. Computers*, vol. 49, no. 9, pp. 967–970, 2000.